#2

대 한 민 국 특 허 청
KOREAN INDUSTRIAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Industrial Property Office.

출 원 번 호 : 특허출원 1999년 제 50627 호
Application Number

출 원 년 월 일 : 1999년 11월 15일
Date of Application

출 원 인 : 삼성전자 주식회사
Applicant(s)

CERTIFIED COPY OF PRIORITY DOCUMENT

2000 년 03 월 16 일

특 허 청
COMMISSIONER

【서류명】 특허출원서
【권리구분】 특허
【수신처】 특허청장
【제출일자】 1999.11.15
【발명의 명칭】 어드레스 트레이스를 이용한 분기 예측 방법
【발명의 영문명칭】 A BRANCH PREDICTION METHOD USING ADDRESS TRACE
【출원인】
【명칭】 삼성전자 주식회사
【출원인코드】 1-1998-104271-3
【대리인】
【성명】 임창현
【대리인코드】 9-1998-000386-5
【포괄위임등록번호】 1999-007368-2
【대리인】
【성명】 권혁수
【대리인코드】 9-1999-000370-4
【포괄위임등록번호】 1999-056971-6
【발명자】
【성명의 국문표기】 박성배
【성명의 영문표기】 PARK,SUNG BAE
【주민등록번호】 580812-1002317
【우편번호】 435-040
【주소】 경기도 군포시 산본동 삼성장미APT 1132동 1504호
【국적】 KR
【심사청구】 청구
【취지】 특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정에 의한 출원심사 를 청구합니다. 대리인
임창현 (인) 대리인
권혁수 (인)
【수수료】
【기본출원료】 20 면 29,000 원
【가산출원료】 0 면 0 원

【우선권주장료】	0 건	0	원
【심사청구료】	4 항	237,000	원
【합계】	266,000 원		
【첨부서류】	1. 요약서·명세서(도면)_1통		

【요약서】

【요약】

여기에 개시된 어드레스 트레이스를 이용한 분기 예측 방법은, 수행되는 명령어에 대응되는 어드레스 트레이스 자체를 디코딩 된 형태로 저장한다. 상기 분기 예측 방법은, 반복되는 루틴의 시작 어드레스와 종료 어드레스를 지정한 후, 상기 루틴이 현재 억세스 되고 있는 횟수와 전체 억세스 횟수를 비교함으로써 상기 루틴의 종료를 확인하고, 다음에 수행될 루틴의 어드레스 정보를 저장한다. 이와 같은 방법에 의해서, 반복되어 수행되는 루틴에 대한 억세스 정보를 적은 용량의 트레이스 캐쉬를 이용하여 저장할 수 있다.

【대표도】

도 5

【명세서】

【발명의 명칭】

어드레스 트레이스를 이용한 분기 예측 방법{A BRANCH PREDICTION METHOD USING ADDRESS TRACE}

【도면의 간단한 설명】

도 1은 4 세대의 마이크로 아키텍처를 보여주기 위한 도면;

도 2a는 일반적인 명령어 캐쉬 내에 저장된 기본 블록들의 동적 시퀀스를 보여주기 위한 도면;

도 2b는 일반적인 트레이스 캐쉬를 보여주기 위한 도면;

도 3은 반복되는 명령어 패턴의 일례를 보여주기 위한 도면;

도 4는 도 3에 도시된 명령어들이 종래 기술에 의한 트레이스 캐쉬에 저장된 패턴을 보여주기 위한 도면;

도 5는 본 발명에 의한 어드레스 트레이스 캐쉬의 구성을 보여주기 위한 도면; 그리고

도 6은 도 3에 도시된 명령어들이 본 발명에 의한 트레이스 캐쉬에 저장된 패턴을 보여주기 위한 도면.

도면의 주요 부분에 대한 부호의 설명

22 : 트레이스 캐쉬 220 : 어드레스 트레이스 캐쉬

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<10> 본 발명은 분기 예측 방법에 관한 것으로, 좀 더 구체적으로는 어드레스 트레이스를 이용한 분기 예측 방법에 관한 것이다.

<11> 도 1은 4 세대의 마이크로 아키텍처를 보여주기 위한 도면이다. 상기 도면은 1997년 9월, James E. Smith와 Sriram Vajapeyam에 의해 IEEE Computer, 68-74쪽에 실린 논문 'Trace Processors : Moving to Fourth-Generation Microarchitectures'의 도 1을 참조한 것이다. 도면을 참조하면, (a)는 1940년대에 최초의 디지털 컴퓨터에 적용되어 1960년대 초기까지 사용된 제 1 세대 마이크로 아키텍처인 직렬 프로세서(serial processor)이다. 이 직렬 프로세서는 다음 실행 전에 각각의 명령어(instruction)를 페치(fetch)하고 실행(execute)한다. 도 1의 (b)에 도시된 파이프라인을 사용하는 제 2 세대 마이크로 아키텍처는 1980년대 말까지 고성능 프로세스를 위한 표준으로 사용되었다. 그리고 도 1의 (c) 및 (d)에 도시된 제 3 세대 및 제 4 세대 마이크로 아키텍처는 수퍼스칼라 프로세서들(super scalar processors)을 사용하는 특징을 가진다. 이들은, 1980년대 말 상업적으로 이용 가능한 프로세서들에 우선적으로 나타나기 시작했다. 도 1에 도시된 바와 같이, 다음 세대의 고성능 프로세서들은, 개개의 수퍼스칼라 파이프라인으로 명령어 그룹들을 발송하는, 보다 높은 수준의 제어를 할 수 있는 다중 수퍼스칼라 파이프라인들을 사용할 것이다. 동시에 다수 개의 명령어들을 캐쉬 블록으로 페치하고 상기 명령어들을 병렬 처리하는 수퍼스칼라 프로세서는, 예측 오류에 의해서라기 보다는 파이프라인의 정지(pipeline stall)와 잘못된 명령어를 무효화하는 것에 의해 야기되는

성능의 저하를 겪게 된다. 그러므로, 정확한 분기 예측은 수퍼스칼라 프로세서를 위해 더욱 중요하다.

<12> 앞에서 설명한 바와 같이, 수퍼스칼라 프로세서를 고성능화 하기 위한 핵심 기술은 파이프라인의 사용을 최적화 하는 기술이며, 여기서 가장 중요한 설계상의 요소는 분기(branch)를 제어하는 방법이다. 분기가 이루어질 것인지를 예측하는 데에는 여러 가지 기술들이 사용될 수 있는데, 이들 중 가장 많이 사용되는 방법으로는 이분 분기 예측(bimodal branch prediction)이 있다. 그러나 최근에는 더 많은 분기 히스토리(branch history)를 사용해서 더욱 정확한 예측결과를 보여주고 있는데, 그 중 한 방법은, 각각의 분기 히스토리를 독립적으로 고려하고 반복적인 패턴들의 이점을 이용하는 지역 분기 예측(local branch prediction) 방법이다. 그리고, 다른 하나의 방법으로는, 예측하는데 있어 최근의 모든 분기들의 결합된 히스토리(combined history)를 사용하는 전역 분기 예측(global branch prediction) 방법이다. 이러한 각각의 분기 예측 방법들은 독특한 이점들을 가진다. 예를 들면, 상기 이분 분기 예측 방법은 각각의 분기가 특정 방향으로 강하게 편중된 경우에 적합하고, 지역 분기 예측 방법은 단순한 반복 패턴들의 분기에 적합하다. 그리고 상기 전역 분기 예측 방법은 순차적으로 실행되는 분기들에 의해 실행되는 방향이 상관도가 매우 높을 때 특히 좋은 특성을 가진다. 앞에서 설명한 바와 같은 분기 예측의 방법들은, 1993년 6월, Scott McFarling에 의해 WRL Technical Note TN-36의 1-19 쪽에 실린 논문 'Combining Branch Predictors'에 소개되어 있다. 여기서, WRL(Western Research Laboratory)이란, 1982년 Digital Equipment Corporation에

의해 설립된 컴퓨터 시스템 연구 단체이다. 앞에서 설명한 분기 예측 방법 외에도, 지역 분기 예측 방법과 전역 분기 예측 방법을 결합한 콤바인드 분기 예측(combined branch prediction) 방법 등이 있다.

<13> 상기와 같은 방법을 사용하는 분기 예측기(branch predictor)는, 분기 명령어를 만날 경우 조건 검사 과정의 결과가 나오기 전에 이전의 분기 명령어의 결과를 사용하여 해당 분기 명령어의 조건 검사 결과를 예측한다. 그리고, CPU(Central Processing Unit)는 예측된 조건 검사 결과에 따라 다음의 명령어를 페치하여 수행한다. 그 결과, 파이프라인 방식을 채택함에 의해 빠른 명령어의 페치(fetch)를 필요로 하는 근래의 CPU에서 성능을 저하시키는 파이프라인의 정지(stall)를 없앨 수 있다. 그러나, 만약 분기 예측의 결과가 틀릴 경우에는, 이미 페치하여 수행 중인 명령어들의 진행을 중지시키고 실제적인 다음 명령어를 페치하여 수행해야 한다. 부정확한 예측은 상기 파이프라인이 적절한 명령어들로 다시 채워질 때까지 정지하게 하는 원인이 된다. 이를 잘못된 예측으로 인한 분기 페널티(mispredicted branch penalty)라 한다.

<14> 5-단 파이프라인을 가지는 프로세서는 보통 두 사이클의 분기 페널티를 가지므로, 예를 들어 4-웨이 수퍼스칼라 설계에는 8개의 명령어에 대한 손실이 발생된다. 파이프라인이 확장되면, 한층 더 많은 명령어들의 손실이 발생함과 동시에 상기와 같은 분기 페널티가 증가하게 된다. 일반적으로, 프로그램들은 4 내지 6 명령어마다 분기를 하게 되기 때문에, 부정확한 분기 예측은 하이 수퍼스칼라 또는 깊은 파이프라인 설계에 있어서 격심한 성능의 저하를 초래한다.

<15> 상기와 같은 분기 페널티를 줄이기 위해 많은 노력들이 이루어지고 있으며, 최근에는 트레이스 캐쉬(trace cache)를 사용하는 트레이스 프로세서가 사용되고 있다. 이와

같은 트레이스 프로세서는 James E. Smith와 Sriram Vajapeyam에 의해 발표된 상기 논문 'Trace Processors : Moving to Fourth-Generation Microarchitectures'에 개시되어 있다.

<16> 도 2a는 일반적인 명령어 캐쉬(instruction cache ; 21) 내에 저장된 기본 블록들의 동적 시퀀스(dynamic sequence)를 보여주기 위한 도면이고, 도 2b는 일반적인 트레이스 캐쉬(22)를 보여주기 위한 도면이다. 도 2a를 참조하면, 도면에 표시된 화살표는 'taken' 분기(분기 목적지 어드레스(branch target address)로 점프하는 경우)들을 나타낸다. 명령어 캐쉬(21)에서는 명령어들이 불연속적인 캐쉬 장소들에 저장되기 때문에, 매 사이클마다 발생되는 다중 분기 예측들조차도 기본 블록들 'ABCDE' 내에서의 명령어들을 페치하기 위해서는 4 사이클이 요구된다.

<17> 이러한 이유 때문에, 몇몇 연구자들은 길이가 긴 동적 명령어 시퀀스들을 캡쳐하기 위한 특정 명령어 캐쉬를 제안해 오고 있다. 이와 같은 캐쉬는, 도 2b에 도시된 바와 같이 각각의 라인이 동적 명령어 스트림(dynamic instruction stream)의 단편(snapshot) 또는 트레이스(trace)를 저장한다. 따라서, 이와 같은 구조를 가지는 캐쉬를 트레이스 캐쉬(22)라 한다. 상기 트레이스 캐쉬(22)는, 1994년 6월 J. Johnson에 의해 Technical Report CSL-TR-94-630, Computer Science Laboratory, Stanford Univ.에 실린 논문 ' Expansion Caches for Superscalar Processors'; 1995년 1월 A. Peleg와 U. Weiser에 의해 취득된 U. S. Pat. No. 5,381,533, 'Dynamic flow instruction cache memory organized around trace segments independant of virtual address line'; 그리고 1996년 12월 E. Rotenberg, S. Bennett 및 J. Smith에 의해 Proc. 29th Int'l Symp. Microarchitecture의 24-34쪽에 실린 논문 'Trace cache : A Low Latency Approach to

High Bandwidth Instruction Fetching'에 각각 개시되어 있다.

<18> 이 외에도 상기 트레이스 캐쉬(22)는, 1998년 6월 Sanjay Jeram Patel, Marius Evers 및 Yale N. Patt에 의해서 Proc. 25th Inter'l Symp. Computer Architecture의 262-271쪽에 발표된 논문 'Improving Trace Cache Effectiveness with Branch Promotion and Trace Packing'; 1999년 2월 Sanjay Jeram Patel, Daniel Holmes Friendly 및 Yale N. Patt에 의해서 IEEE TRANSACTIONS ON COMPUTER, VOL. 48, NO.2의 193-204쪽에 발표된 논문 'Evaluation of Design Options for the Trace Cache Fetch Mechanism'; 그리고 1999년 2월 Eric Rotenberg, Steve Bennett 및 James E. Smith에 의해서 IEEE TRANSACTIONS ON COMPUTER, VOL. 48, NO.2의 111-120쪽에 발표된 논문 'A Trace Cache Microarchitecture and Evaluation'에 개시되어 있다.

<19> 앞에서 설명한 바와 같이, 도 2a에 도시된 명령어 캐쉬(21) 내에 불연속적으로 나타난 블록들과 동일한 동적 시퀀스는, 도 2b에 도시된 트레이스 캐쉬(22) 내에 연속적으로 나타내진다. 따라서, 기존의 분기 예측 방법에서와 같이 프로그램된 루틴에 따라 명령어가 있는 어드레스로 반복적인 분기를 수행하지 않고도 트레이스 캐쉬(22)에 저장된 명령어들을 순차적으로 실행할 수 있다. 따라서, 기존의 분기 예측 방법에서 초래되는 분기 패널티를 방지할 수 있을 뿐만 아니라, 명령어 캐쉬(21)의 불연속적인 위치에 저장되는 명령어들을 트레이스 캐쉬(22) 내에 연속적으로 저장함으로써 보다 진보된 병렬 처리를 수행할 수 있다.

<20> 그러나 상기와 같은 트레이스 캐쉬(22)는, 명령어 자체를 저장하기 때문에 이 명령에 대응되는 어드레스로의 디코딩 과정이 필요하다. 그리고, 상기와 같은 트레이스

캐쉬(22)는, 반복적으로 실행되는 명령어들조차도 실행되는 순서에 따라 되풀이해서 저장하기 때문에 캐쉬 사이즈가 너무 커지는 단점이 있다. 따라서, 상기와 같은 방식으로 명령어들을 저장하는 트레이스 캐쉬(22)가 모든 명령어를 저장할 만큼 충분한 크기를 갖기 위해서는, 칩 사이즈 및 생산 단가가 증가하는 문제가 발생된다. 따라서, 트레이스 캐쉬(22)의 어드레스 디코딩 시간을 줄일 수 있고, 적정량의 캐쉬 용량을 사용하여 칩 사이즈 및 생산 단가를 줄일 수 있는 분기 예측 방법이 요구된다.

【발명이 이루고자 하는 기술적 과제】

<21> 따라서, 본 발명의 목적은 상술한 제반 문제점을 해결하기 위해 제안된 것으로, 트레이스 캐쉬를 사용하는 분기 예측 방법에 있어서, 어드레스 디코딩 시간을 줄일 수 있고, 적은 용량의 트레이스 캐쉬로 정확한 분기 예측을 수행함으로써 칩 사이즈 및 생산 단가를 줄일 수 있는 분기 예측 방법을 제공하는데 있다.

【발명의 구성 및 작용】

<22> 상술한 바와 같은 본 발명의 목적을 달성하기 위한 본 발명의 특징에 의하면, 트레이스 캐쉬를 사용하는 분기 예측 방법은, 반복되지 않는 명령어들로 이루어진 루틴이 실행되는 경우 실행되는 명령의 순서에 따라 각각의 명령어에 대응되는 어드레스를 상기 트레이스 캐쉬에 저장하는 단계, 그리고, 반복되는 명령어들로 이루어진 루틴이 실행되는 경우 상기 루틴이 시작되는 어드레스, 상기 루틴이 종료되는 어드레스, 그리고 상기 루틴의 현재 억세스 횟수 및 상기 루틴의 전체 억세스 횟수를 카운트하여 저장하는 단계를 포함한다.

<23> 상기 트레이스 캐쉬는, 반복되는 명령어들로 이루어진 상기 루틴이 실행되는 경우,

상기 루틴의 현재 억세스 횟수 및 상기 루틴의 전체 억세스 횟수를 카운트하기 위한 루프 카운터들을 포함한다.

<24> 상기 분기 예측 방법은, 상기 루프 카운터들에 의해 카운트된 계수들을 비교하여, 상기 두 계수가 서로 같을 때 상기 루틴 다음에 실행될 루틴의 시작 어드레스를 어드레싱하는 단계를 포함한다. 그리고, 잘못된 분기 예측이 발생된 경우 상기 루프 카운터를 재구성하는 단계를 포함하되, 상기 루프 카운터는 가장 최근에 업데이트 된 루프 카운트를 사용한다.

<25> (실시예)

<26> 이하 본 발명에 따른 실시예를 첨부된 도면 도 3 내지 도 6을 참조하여 상세히 설명한다.

<27> 본 발명의 신규한 트레이스 캐쉬는, 명령어에 대응되는 어드레스 트레이스 자체를 디코딩 된 형태로 저장하므로 각각의 명령어에 대한 어드레스 디코딩 시간을 줄일 수 있다. 그리고, 반복 수행되는 루틴에 대한 어드레스 트레이스의 저장시 적은 용량의 트레이스 캐쉬를 사용하므로 칩 사이즈 및 생산 단가를 줄일 수 있다.

<28> 도 3은 반복되는 명령어 패턴의 일례를 보여주기 위한 도면이다.

<29> 먼저, 도 3에 도시된 루틴 ①을 참조하면, 연산 A 및 연산 B가 30회 반복해서 수행된다. 이와 같은 루틴 ①의 수행이 종료되면, 루틴 ②가 실행되는데, 루틴 ②에서는 연산 C, 연산 D 및 연산 E가 순차적으로 20회 반복 수행된다. 그리고, 이와 같은 루틴 ②의 수행이 종료되면, 연산 F 및 연산 G가 40회 반복 수행되는 루틴 ③이 수행된다.

<30> 예를 들어, 도 3에 도시된 바와 같은 루틴들이 수행된다고 가정할 때, 종래 기술에

의한 트레이스 캐쉬(22)에 저장되는 명령어들은 도 4와 같다.

<31> 도 4를 참조하면, 종래의 트레이스 캐쉬(22)는, 실행되는 명령어들이 반복적으로 실행되거나 또는 반복적으로 실행되지 않거나 상관하지 않고, 단지 실행되는 순서에 따라 명령어들을 저장한다. 따라서, 상기 트레이스 캐쉬(22)는, 루틴 ①을 위한 60개의 명령어(명령어 2개 ×30회 반복 = 60)를 저장하기 위한 데이터 저장 영역과, 루틴 ②를 위한 60개의 명령어(명령어 3개 ×20회 반복 = 60)를 저장하기 위한 데이터 저장 영역, 그리고 루틴 ③을 위한 80개의 명령어(명령어 2개 ×40회 반복 = 80)를 저장하기 위한 데이터 저장 영역이 각각 요구된다. 즉, 도 3에 도시된 루틴 ① 내지 루틴 ③을 저장하기 위해서는; 총 200개의 데이터 저장 영역이 요구된다. 이 경우, 각각의 명령어들을 저장하는데 32 bits가 소요된다면, 상기 루틴 ① 내지 루틴 ③을 저장하기 위해서는 총 6,400 bits(즉, 800 Bytes)의 데이터 저장 영역이 필요로 하게된다.

<32> 도 5는 본 발명에 의한 어드레스 트레이스 캐쉬(220)의 구성을 보여주기 위한 도면이다. 본 발명에 의한 어드레스 트레이스 캐쉬(220)는, 도 5에 도시된 바와 같이, 각각의 루틴이 시작되는 어드레스를 저장하기 위한 스타트 어드레스(start address)와, 각각의 루틴이 종료되는 어드레스를 나타내기 위한 엔드 어드레스(end address), 해당 루틴의 현재 억세스 횟수를 카운트하기 위한 커런트 억세스 루프 카운터(current access loop counter), 그리고 상기 루틴의 전체 억세스 횟수를 나타내기 위한 올드 억세스 루프 카운터(old access loop counter)로 구성된다.

<33> 예를 들어, 도 5에 도시된 루틴 ①에서 수행되는 명령어들의 수행 정보를 본 발명에 의한 어드레스 트레이스 캐쉬(220)로 나타내는 경우, 먼저 루틴 ①의 스타트 어드레스 및 엔드 어드레스가 상기 트레이스 캐쉬(220)에 저장된다. 이어서, 상기 커런트 억세

스 루프 카운터에는 루틴 ①의 현재 억세스 횟수가 저장되고, 올드 억세스 루프 카운터에는 루틴 ①의 전체 억세스 횟수(예를 들면, 30회)가 저장된다. 루틴 ①의 억세스가 반복됨에 따라 상기 커런트 억세스 루프 카운터의 값이 증가하게 되는데, 만약 상기 커런트 억세스 루프 카운터의 값이 올드 억세스 루프 카운터의 값과 같게 되면 루틴 ①의 수행이 종료되고, 루틴 ②의 스타트 어드레스가 NFP(next fetch point)로서 저장된다.

<34> 도 3 및 도 5에 도시된 바와 같이, 연이어 수행되는 루틴 ① 내지 루틴 ③은, 앞에서 설명한 바와 같은 방법에 의해 상기 어드레스 트레이스 캐쉬(220)에 각각 저장될 수 있다. 그러나, 상기 루틴 ① 내지 루틴 ③과 같이 반복되어 실행되지 않는 루틴의 경우에는, 그 루틴을 이루고 있는 명령어 각각에 대한 어드레스를 순차적으로 기입한다. 그리고, 잘못된 분기 예측이 발생된 경우에는 상기 루프 카운터를 재구성하는데, 여기에는 가장 최근에 업데이트 된 루프 카운트가 사용된다.

<35> 도 6은 도 3에 도시된 명령어들이 본 발명에 의한 어드레스 트레이스 캐쉬(220)에 저장된 패턴을 보여주기 위한 도면이다.

<36> 도 6을 참조하면, 예를 들어 도 3에 도시된 루틴 ①을 본 발명에 의한 어드레스 트레이스 캐쉬(220)에 저장하는 경우, 루틴 ①의 스타트 어드레스로는 최초에 실행되는 명령어 A의 어드레스가 저장되고, 루틴 ①의 엔드 어드레스로는 최후에 실행되는 명령어 B의 어드레스가 각각 저장된다. 루틴 ①의 경우, 전체 반복되는 횟수는 30회이므로, 올드 억세스 루프 카운터는 30으로 저장되고, 루틴 ①이 반복해서 실행될 때마다 커런트 억세스 루프 카운터 값이 1 씩 증가하게 된다. 이와 동일한 방법으로 루틴 ② 및 루틴 ③에 대한 정보가 상기 어드레스 캐쉬(220)에 각각 저장된다.

<37> 도 6에 도시된 바와 같이, 본 발명에 의한 어드레스 캐쉬(220)는 각각의 루틴이 시

작되는 어드레스와, 각각의 루틴이 종료되는 어드레스, 커런트 억세스 루프 카운터 및 올드 억세스 루프 카운터로 구성되기 때문에, 반복되는 루틴에 대한 정보를 저장하기 위해서는 단지 4개의 데이터 저장 영역이 요구된다. 따라서, 본 발명에 의한 어드레스 트레이스 캐쉬(220)로 도 3에 도시된 상기 루틴 ① 내지 루틴 ③을 저장하기 위해서는 총 12개의 데이터 저장 영역이 요구된다. 이 경우, 각각의 정보를 저장하는데 32 bits가 소요된다면, 상기 루틴 ① 내지 루틴 ③을 저장하기 위해서는 총 384 bits(즉, 48 Bytes)가 요구된다. 이는 도 4에 도시된 종래 기술에 의한 트레이스 캐쉬에 비해 소요되는 데이터 저장 영역이 약 16.7배 가량 줄어든 것이다.

<38> 이와 같은 효과는, 반복되는 명령이 많이 포함된 루틴일수록, 그리고 명령어들의 반복되는 횟수가 많을수록 더욱 커진다. 이와 같이, 본 발명에 의한 어드레스 트레이스 캐쉬(220)는 종래의 트레이스 캐쉬에 비해 현저히 줄어든 데이터 저장 영역을 사용하기 때문에, 칩 사이즈 및 생산 단가를 줄일 수 있다. 뿐만 아니라, 본 발명에 의한 어드레스 트레이스 캐쉬(220)는 각각의 명령어에 대한 어드레스 트레이스 자체를 디코딩된 형태로 저장하므로, 명령어의 어드레스 디코딩 시간을 줄일 수 있다.

<39> 이상에서, 본 발명에 따른 회로의 구성 및 동작을 상기한 설명 및 도면에 따라 도시하였지만 이는 예를 들어 설명한 것에 불과하며 본 발명의 기술적 사상을 벗어나지 않는 범위 내에서 다양한 변화 및 변경이 가능함은 물론이다.

【발명의 효과】

<40> 이상과 같은 본 발명에 의하면, 명령어에 대응되는 어드레스 트레이스 자체를 디코딩 된 형태로 저장하므로 각각의 명령어에 대한 어드레스 디코딩 시간을 줄일 수 있고,

적은 용량의 트레이스 캐쉬를 사용하므로 칩 사이즈 및 생산 단가를 줄일 수 있다.

【특허청구범위】

【청구항 1】

트레이스 캐쉬를 사용하는 분기 예측 방법에 있어서:

반복되지 않는 명령어들로 이루어진 루틴이 실행되는 경우, 실행되는 명령의 순서에 따라 각각의 명령어에 대응되는 어드레스를 상기 트레이스 캐쉬에 저장하는 단계; 그리고,

반복되는 명령어들로 이루어진 루틴이 실행되는 경우, 상기 루틴이 시작되는 어드레스, 상기 루틴이 종료되는 어드레스, 그리고 상기 루틴의 현재 억세스 횟수 및 상기 루틴의 전체 억세스 횟수를 카운트하여 저장하는 단계를 포함하는 것을 특징으로 하는 어드레스 트레이스를 이용한 분기 예측 방법.

【청구항 2】

제 1 항에 있어서,

상기 트레이스 캐쉬는,

반복되는 명령어들로 이루어진 상기 루틴이 실행되는 경우,

상기 루틴의 현재 억세스 횟수 및 상기 루틴의 전체 억세스 횟수를 카운트하기 위한 루프 카운터들을 포함하는 것을 특징으로 하는 어드레스 트레이스를 이용한 분기 예측 방법.

【청구항 3】

제 2 항에 있어서,

상기 분기 예측 방법은,

상기 루프 카운터들에 의해 카운트된 계수들을 비교하여, 상기 두 계수가 서로 같을 때 상기 루틴 다음에 실행될 루틴의 시작 어드레스를 어드레싱하는 단계를 포함하는 것을 특징으로 하는 어드레스 트레이스를 이용한 분기 예측 방법.

【청구항 4】

제 2 항에 있어서,

상기 분기 예측 방법은,

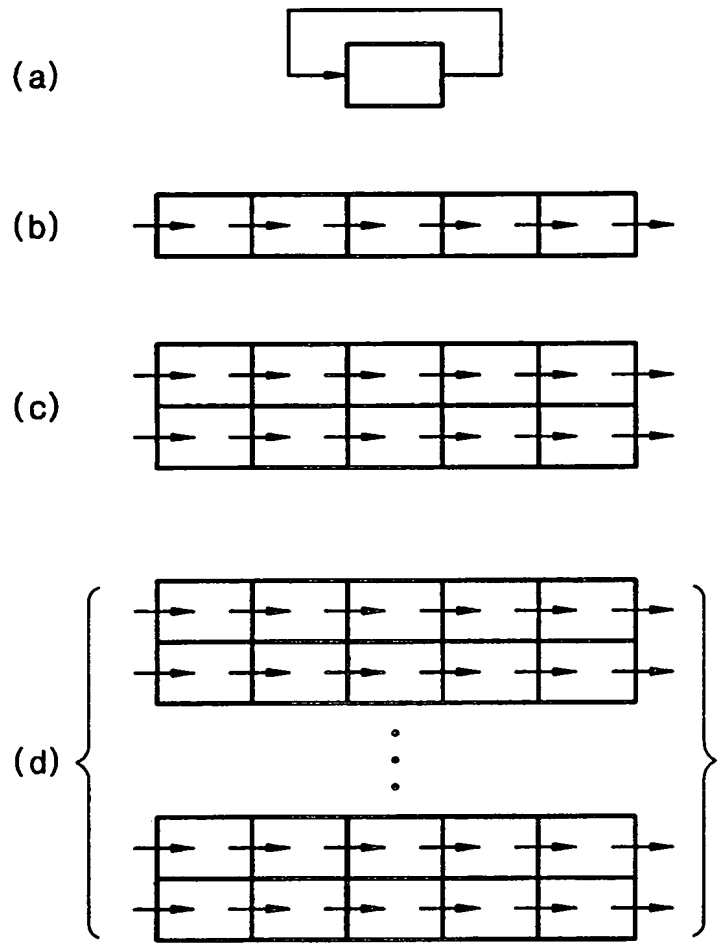
잘못된 분기 예측이 발생된 경우, 상기 루프 카운터를 재구성하는 단계를 포함하되

,

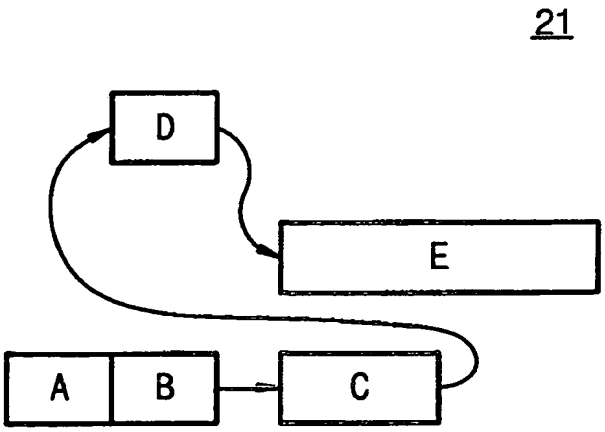
상기 루프 카운터는 가장 최근에 업데이트 된 루프 카운트를 사용하는 것을 특징으로 하는 어드레스 트레이스를 이용한 분기 예측 방법.

【도면】

【도 1】

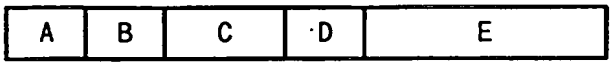


【도 2a】



【도 2b】

22

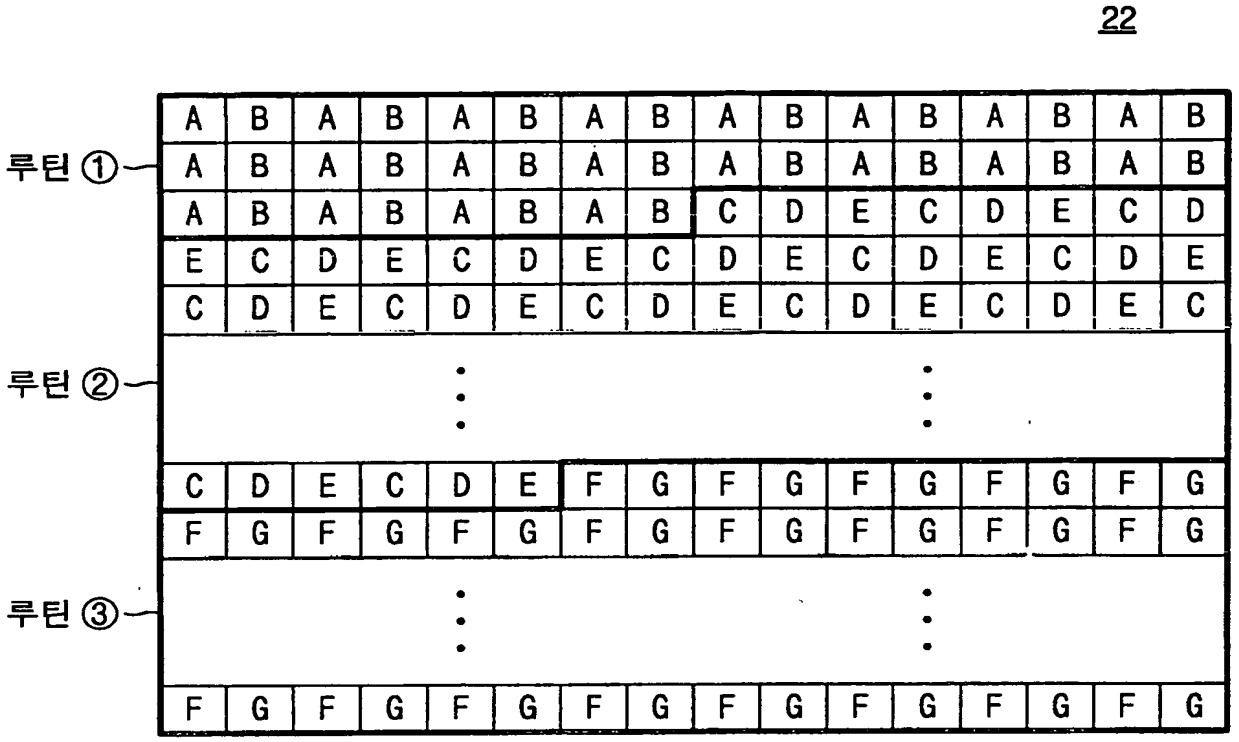


【도 3】

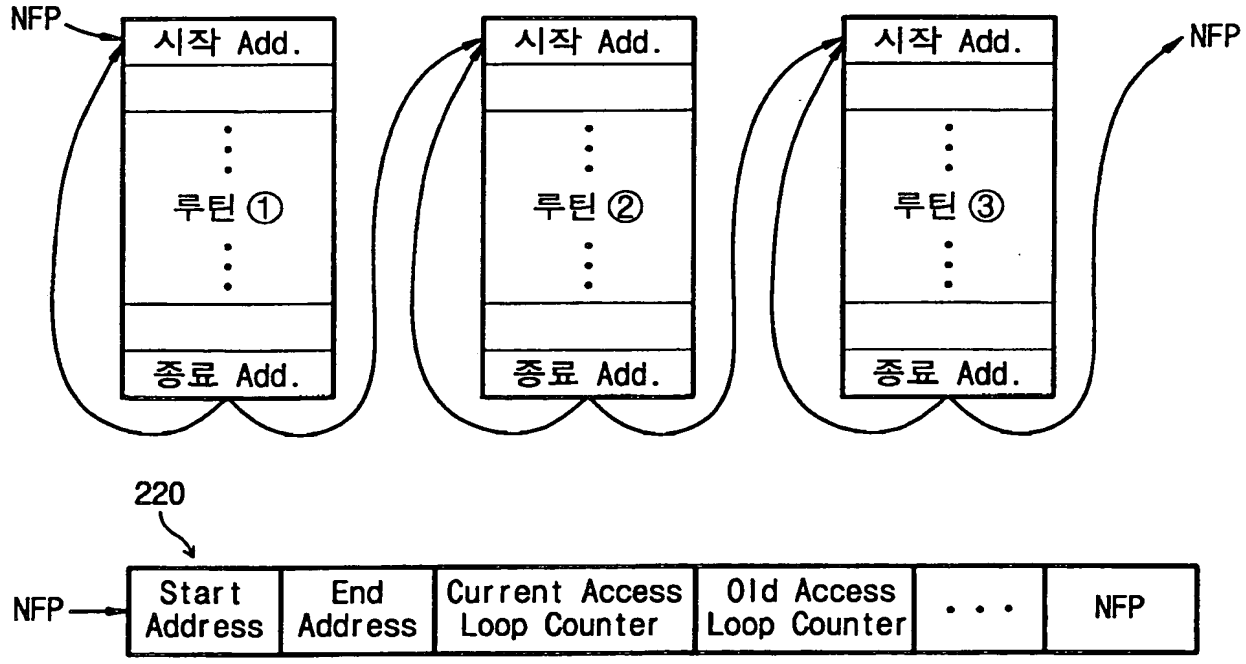
```
for (i=1 ; i≤30 ; i++)
{
      연산 A ;
      연산 B ;
}                                  } 루틴 ①

for (j=1 ; j≤20 ; j++)
{
      연산 C ;
      연산 D ;
      연산 E ;
}                                  } 루틴 ②

for (k=1 ; k≤40 ; k++)
{
      연산 F ;
      연산 G ;
}                                  } 루틴 ③
```

【도 4】

(종래 기술)


22
루틴 ①
루틴 ②
루틴 ③

【도 5】


NFP
시작 Add.
루틴 ①
종료 Add.
시작 Add.
루틴 ②
종료 Add.
시작 Add.
루틴 ③
종료 Add.
NFP
220
NFP
Start Address
End Address
Current Access Loop Counter
Old Access Loop Counter
NFP

【도 6】

220